

TM No. 941106  
REFERENCE COPY

NAVAL UNDERSEA WARFARE CENTER  
DETACHMENT NEW LONDON  
NEW LONDON, CONNECTICUT 06320-5594

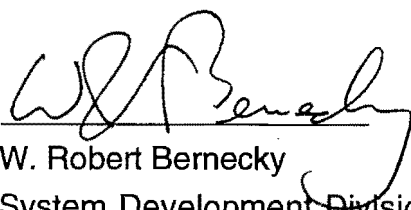


Technical Memorandum

**Multi-SIB iWarp Array  
Configuration**

Date: 22 August 1994

Prepared by:

  
W. Robert Bernecky  
System Development Division  
Submarine Sonar Department

Approved for public release; distribution unlimited.

Report Documentation Page			Form Approved OMB No. 0704-0188		
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE <b>22 AUG 1994</b>		2. REPORT TYPE <b>Technical Memorandum</b>		3. DATES COVERED <b>22-08-1994 to 22-08-1994</b>	
4. TITLE AND SUBTITLE <b>Multi-SIB iWarp Array Configuration</b>			5a. CONTRACT NUMBER		
			5b. GRANT NUMBER		
			5c. PROGRAM ELEMENT NUMBER		
6. AUTHOR(S) <b>W. Bernecky</b>			5d. PROJECT NUMBER		
			5e. TASK NUMBER		
			5f. WORK UNIT NUMBER		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) <b>Naval Undersea Warfare Center Detachment New London, New London, CT, 06320</b>			8. PERFORMING ORGANIZATION REPORT NUMBER <b>TM 941106</b>		
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) <b>Office of Naval Research Computer Technology Block</b>			10. SPONSOR/MONITOR'S ACRONYM(S)		
			11. SPONSOR/MONITOR'S REPORT NUMBER(S)		
12. DISTRIBUTION/AVAILABILITY STATEMENT <b>Approved for public release; distribution unlimited</b>					
13. SUPPLEMENTARY NOTES <b>NUWC2015</b>					
14. ABSTRACT <b>This document describes how to incorporate multiple SUN Interface Boards into an Intel iWarp parallel processing array. The described procedure was originally developed at Carnegie-Mellon University, and was applied to a 64-cell iWarp array at NUWC/NL. The addition of multiple SIBs significantly improves the input/output capacity of the array.</b>					
15. SUBJECT TERMS <b>iWarp; Intel; 64-cell; Advanced Sonar Processing Architectures; ASPA</b>					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT <b>Same as Report (SAR)</b>	18. NUMBER OF PAGES <b>15</b>	19a. NAME OF RESPONSIBLE PERSON
a. REPORT <b>unclassified</b>	b. ABSTRACT <b>unclassified</b>	c. THIS PAGE <b>unclassified</b>			

## **ABSTRACT**

This document describes how to incorporate multiple SUN Interface Boards into an Intel iWarp parallel processing array. The described procedure was originally developed at Carnegie-Mellon University, and was applied to a 64-cell iWarp array at NUWC/NL. The addition of multiple SIBs significantly improves the input/output capacity of the array.

## **ADMINISTRATIVE INFORMATION**

The research described in this memorandum was performed under the Advanced Sonar Processing Architectures (ASPA) program, Principal Investigator Dr. J. Munoz. The sponsoring activity is the Office of Naval Research (ONR), Computer Technology Block, Program Manager Elizabeth Wald.

The author of this memorandum is located at the Naval Undersea Warfare Center, Detachment , New London CT 06320-5594. The technical reviewer for this document was Gilbert Bowman.

## **ACKNOWLEDGMENTS**

The author wishes to acknowledge the very appreciated expertise of Gilbert Bowman (Code 2123), Thomas Warfel (Carnegie-Mellon University) and Wire Moore (Intel Corp.).

**TABLE OF CONTENTS**

Introduction.....4

Modifying the iWarp .....4

1. Hardware clock source and distribution.....4

2. SIB initialization.....5

3. RTS Backbone Routing .....6

4. Configuration File Syntax.....7

5. RTS Modifications.....8

6. VME Master Interface Programming .....9

Physical Connection of the Second SIB ..... 11

Attaching the EIB boards to the backplane .....12

Booting the Array/Using the aux-SIB ..... 13

Diagnostics .....13

References .....14

## Introduction

The iWarp is a parallel processor, consisting of a number of individual processors (cells) connected by a mesh communication network. A typically-sized system has 64 cells organized in an 8 row x 8 column *array*. Each iWarp array also has a SUN Interface Board (SIB) that serves as the interface between a SUN workstation (known as the host) and the iWarp array. In this capacity, the SIB must sit on the VMEbus of the SUN host. The SIB has, in addition to an iWarp processor chip, hardware to support communication over the VMEbus. In particular, the SIB supports both a VMEbus master interface, and dual-ported memory (a slave interface).

The SIB is generally the only available mechanism by which data can be transferred into and out of the array. Thus, additional SIBs improve the array's input/output capacity. Auxiliary SIBs (aux-SIBs) can also serve as interfaces to VMEbus compatible devices other than SUN workstations. Note that aux-SIBs are meant to sit on a VMEbus different from that of the SUN host workstation.

This document describes the necessary steps required to accommodate multiple SIBs on a single array. This process was originally developed by Thomas Warfel (thomas.warfel@cs.cmu.edu), as a graduate student at Carnegie-Mellon University.

## Modifying the iWarp

The basic idea behind configuring a multi-SIB system is to make the auxiliary SIB(s) look like regular array cells to the runtime system (RTS) so that it can boot them. From there, the RTS can be used to develop and execute the aux-SIB interface program to meet the user's needs.

The major configuration concerns with a multi-SIB system are:

- SIB hardware clock source and distribution
- SIB initialization
- Backbone (RTS) communications snake routing.
- Configuration file syntax
- RTS modifications to support the aux-SIB
- Master interface programming

### 1. Hardware clock source and distribution

The aux-SIB must not export its clock signal to the rest of the array. There are jumpers provided on the SIB that disable the spreading its clock to other cells. The jumper configuration is:

*E10-11    External Clock via pathway*  
*E13-14    SIB is a slave in the system.*

These must be selected on the auxiliary SIBs.

The global registers are a set of registers on the SIB that are accessible by VMEbus masters. The location of these registers is determined by two switches on the edge of the SIB board, and has a default value of 0x7000 in the short address space of the VMEbus. The Global Control Register GCR, address 0x7006, must have its EXTCLK bit set by the VMEbus master. Note that because no program can be down-loaded to the SIB via the array at this point, a separate computer must perform this initialization of the SIB's GCR.

The SIB uses the on-board clock for initialization and for stand alone operation. When a SIB is communicating with an iWarp system or a Single Board Array configuration, the external clock *must* be used.

*The EXTCLK is set by writing a 0x0090 to location 0x7006.*

It is convenient at the same time for the VMEbus master to initialize the address of the dual-ported SIB memory, by writing, e.g., 0x1000 to the Global Slave Address Register GSAR at location 0x7004.

## 2. SIB initialization

Auxiliary SIBs need special handling during reset. An auxiliary SIB can be reset in any of three ways:

- during power up,
- through the reset line on the X pathways as a member of an array, and
- through a bit in the Global Control Register (GCR) on the VME-side.

The *Hold/Reset jumper* should be changed from its default E6-E7 to E7-E8. This enables an automatic clear of the GRSTSIB signal, and so prevents the SIB from locking up if the GRSTSIB is set by mistake.

When reset, the SIB will run code from either the on-chip ROM or the on-board EEPROM, depending on the hardware configuration. The on-chip ROM consists of chip initialization code and a command interpreter (ROM-CI) that awaits a boot message from the pathways. The EEPROM consists of board initialization code and a Diagnostic Array Kernel (DAK) that awaits a boot message from the dual-ported RAM (DPRAM) via the VMEbus.

Unlike the host-SIB, which only uses the DAK when booted by the SUN, an auxiliary SIB must execute both the DAK and the ROM-CI. It needs the DAK to initialize the board logic (Xylinx parts) and it needs the ROM-CI to get bootstrapped into the array by the RTS software, because aux-SIBs have no host.

The decision to run the ROM-CI or the DAK is made by the iWarp chip based on the state of the Stat0 line (It appears as the Stat0 bit in the iWarp chip CONSTAT register and is also attached to the green LED on the SIB). This decision can be controlled by disconnecting the pull-down resistor R820D from U821, pin 6 (Stat0), and connecting it to a toggle switch. When closed the DAK is booted, otherwise the ROM-CI is booted. The system is cold-started in DAK-mode (*SIB mode*), and then the switch is flipped (to *cell mode*). The array is now reset (via the X-pathway reset signal) from the host-SIB by starting the *iwmond* or by using *iwreset*. This causes the ROM-CI to run and the SIB to boot the RTS like the other array cells.

By using the GCR on the aux-SIB VMEbus, an aux-SIB can be forced to reset at any time. If this capability is needed, the user must develop the VME-side access mechanisms (e.g., an SIB device driver). Intel can likely provide the sources to its Sun driver for reference.

### 3. RTS Backbone Routing

The RTS establishes a backbone communication pathway that threads its way through the array. It is along this backbone that the RTS downloads user programs, and passes system messages, and file i/o. Figure 1 depicts the backbone routing for a 4 x 4 array. As can be seen, connecting an aux-SIB to the X-pathways has an insignificant impact on the routing of the backbone.

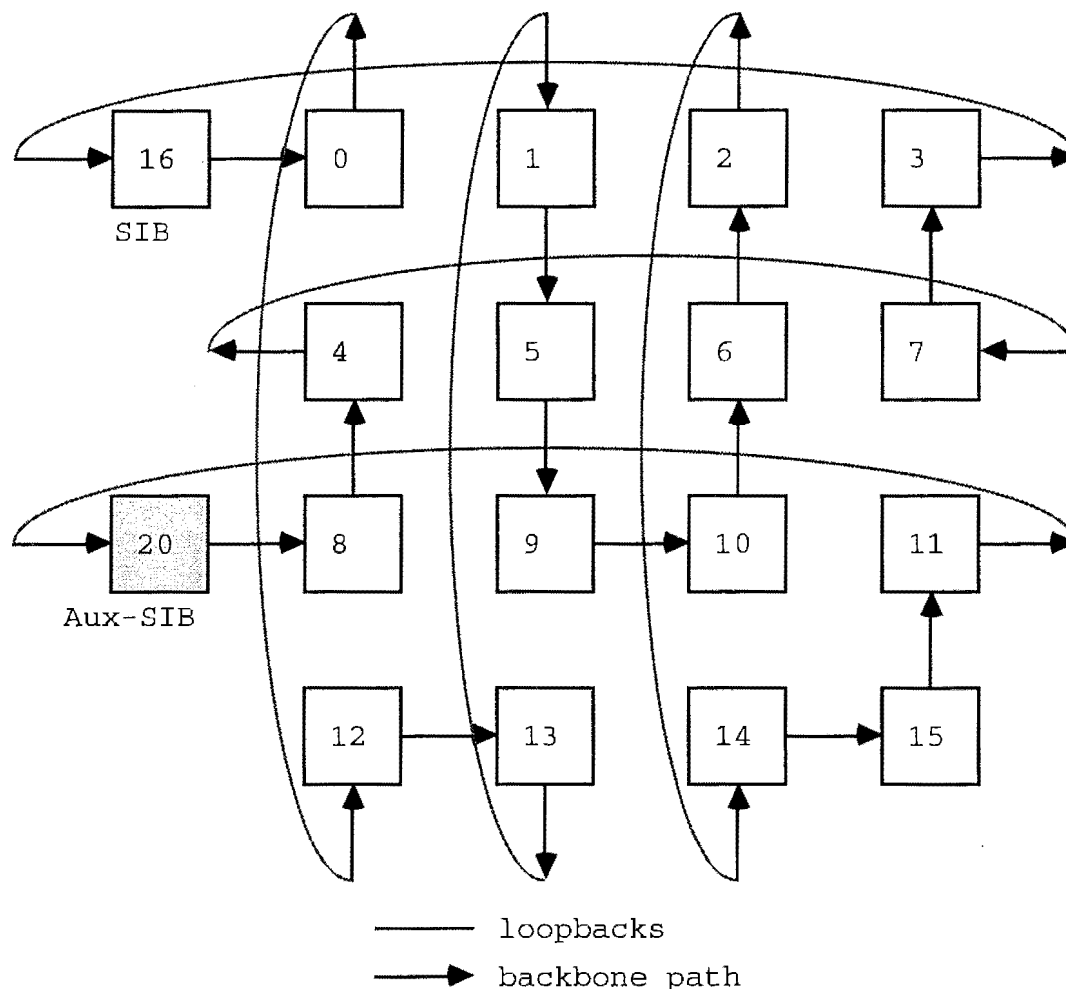


Figure 1. RTS backbone for a 4x4 array with one aux-SIB. The backbone starts at the host-SIB (Cell ID 16).

The aux-SIB may or may not also be connected to a Y pathway. However, connecting the Y pathways greatly complicates the task because the RTS backbone must be configured so that it does not pass through the aux-SIB twice. This necessitates modifying the RTS software.

## 4. Configuration File Syntax

The configuration file should look something like:

```

bcell s*= a but {
    vecpri = 0xc803;
    board="SIB";
    sibtype = 0x403;
    nbrmode = FARU | FARD | FARR | FARL;
    mem={      # 0.5 MB Fast RAM
        0x0..0x7ffff, FAST | RAM | SINGLE;
        0x800000..0x87ffff, SLOW | EPROM | SINGLE;
        0xa00000..0xa3ffff, SLOW | RAM | DUAL;
    };
};

# use this definition for the iWarp slave SIB
t = s but
{
    vecpri = 0xc903;
    board="SBA";
};

```

Place a *t* where the second SIB will be:

```

array z[8,8] = (
    | | | | | | | | ),
    (s a-b-a-b-a-b-a-b),
    ( | | | | | | | | ),
    (- c-d-c-d-c-d-c-d),
    ( | | | | | | | | ),
    (- a-b-a-b-a-b-a-b),
    ( | | | | | | | | ),
    (- c-d-c-d-c-d-c-d),
    ( | | | | | | | | ),
    (- a-b-a-b-a-b-a-b),
    ( | | | | | | | | ),
    (- c-d-c-d-c-d-c-d),
    ( | | | | | | | | ),
    (- a-b-a-b-a-b-a-b),
    ( | | | | | | | | ),
    (- c-d-c-d-c-d-c-d),
    (t a-b-a-b-a-b-a-b),
    ( | | | | | | | | ),
    (- c-d-c-d-c-d-c-d);

```



## 5. RTS Modifications

One small modification needs to be made to the RTS cell kernel to allow the use of the second SIB. Basically, a small amount of code must be commented out to prevent initialization of a host-proxy-interface service on the aux-SIB. Obtaining this modification requires that the user run RTS version 3.2 software on their system.

To connect an aux-SIB's Y pathways, the RTS backbone must be configured so that it does not pass through the aux-SIB twice. This requires modification to the RTS software and so requires access to the RTS build environment.

The file that must be modified is:

```
../iwsys/boot/boot.c
```

There is a makefile for the kernel:

```
../iwsys/IWRTS
```

The run-time system *iwrts* is built by going to the IWRTS directory and issuing the command

```
% make PDE=3.2
```

assuming that the compiler is installed in */iwarp/bin3.2*. Edit the makefile if the compiler is somewhere else. This will generate a kernel, named *iwrts* in the current working directory. No libraries beyond *libc* are required to build the kernel.

Once built, the kernel can be run by placing it in */iwarp/boot* (remember to save the old one first) and restarting *iwmond*. Note that *iwmond* only reads the kernel file once, when it starts.

### boot.c modification

Search for the string "Edge cell, but not" in *../iwsys/boot.c* and modify the code by adding an *ifdef*, as follows:

```

else {
    /*
     * Edge cell, but not *the* SIB
     */
    iep = (imsg_env *)&local_env;
    iep->ie_cellid = _cellid;
    iep->ie_scellid = _scellid;
    iep->ie_hcellid = _hcellid;
    imsg_init_small(iep, 0, 0);
#ifdef 0
    iep = (imsg_env *)0xa00000;
    iep->ie_cellid = _cellid + 1;
    iep->ie_scellid = _scellid;
    iep->ie_hcellid = _hcellid;
#endif
}
sys_init();
}

```

This prevents a host-proxy imsg service from being started on aux-SIBs. This should not be started because there is no "host" on the second SIB.

## 6. VME Master Interface Programming

Intel's RTS does not directly support access to the VME master interface on the host SIB. The master interface has been used successfully at Intel, CMU and NUWC. However, only the small window seems to work. To allow the SIB access to the control registers, be sure to reset the local memory size register on the aux-SIB. Otherwise, any attempt to use e.g., the VME master interface, will result in either a local memory access error on a read, or a never-complete on a write to these registers. The following code segment performs this function:

```
#include <regnums.h>
#include <asm/gen_asm.h>

asm_writecsreg(CSR_LMSIZE, 0x17f);
```

Two types of External Interface Boards: Left and Right. EIBL attaches to XL of a cell e.g. EIBL to Cell 0 and EIBR attaches to XR of cell 7. Only EIB Rights are labelled, (with an R). No label means it is a Left.

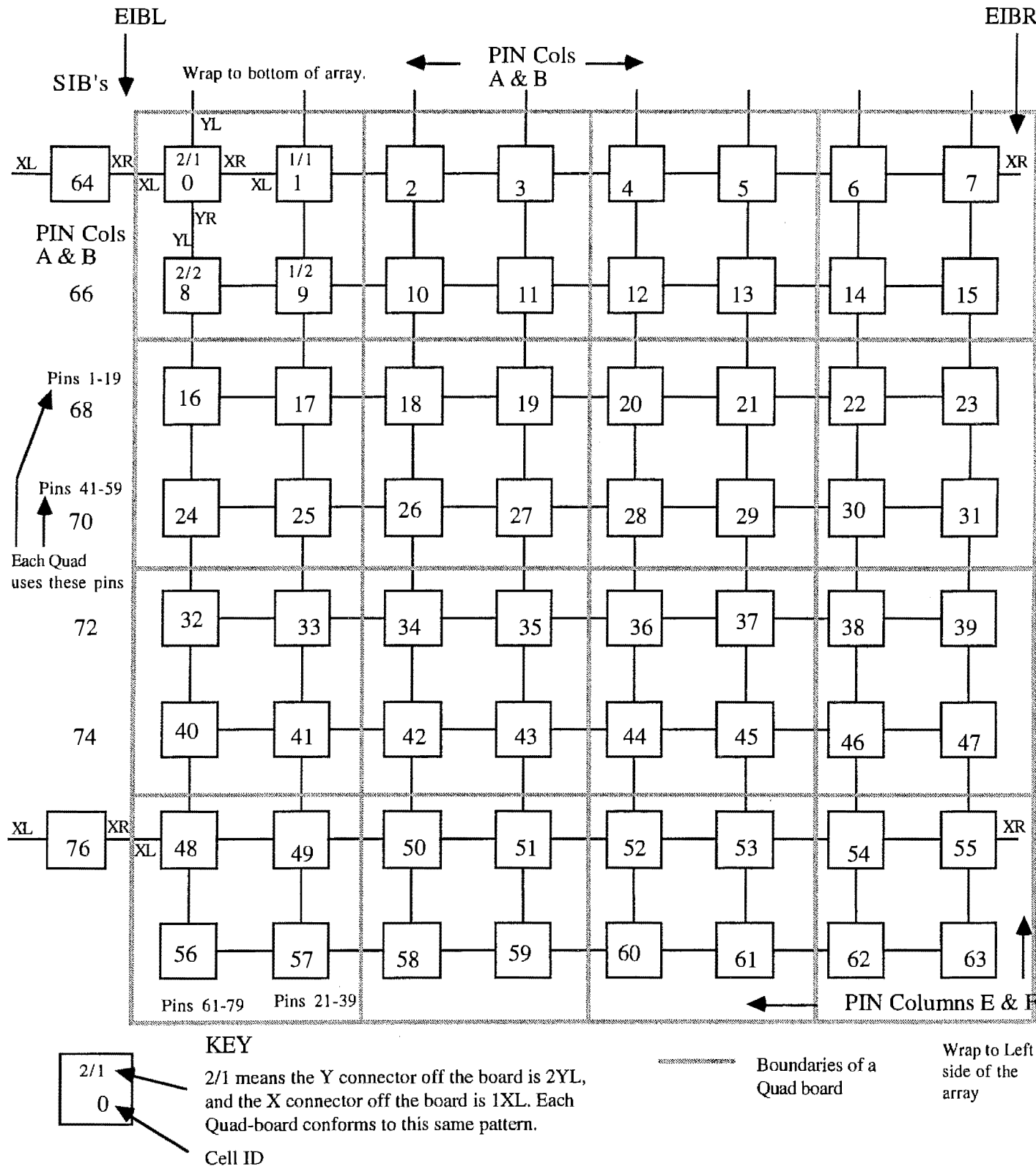


Figure 2

## Physical Connection of the Second SIB

Multiple SIB's may be connected to the array, along the left-hand side of the array, or along the top row. This document specifically addresses the attachment of a SIB to the left-hand side. As shown in Figure 2, the SIB's are numbered according to their position, regardless of the existence of other SIBs. The host SIB must be attached to the first row, and has cell ID 64, i.e., one more than the last cell in the array. Auxiliary SIB ID's are assigned in ascending numbers, incrementing by two. (The second number is to accommodate a potential "host" number.)

Note that an aux-SIB may not be attached to the bottom row of the array.

The SIB is connected to the array by an External Interface Board (EIB) kit. Figure 2 shows that the XR port of the SIB is attached to the XL port of the left-most cell in some row. Similarly, the XL port of the SIB is attached to the XR port of the right-most cell in that same row.

The EIB kit consists of the following items:

1. Two EIB differential driver boards. These come in pairs, Left and Right. The EIB Rights are labeled with an R. The EIB Lefts are not labeled. Each board has a connector on the back that plugs onto the pins of the iWarp backplane. Details on plugging onto the backplane are given in the following section.

There are two connectors on the front of the board, which I will refer to as connector 1 and connector 2. Connector 1 is that one closest to the (front) edge of the board.

2. Two brown EIB cables. An EIB cable has two flat connectors at one end that connect to a differential driver board. The two connectors are distinguished by a *long shank* and a *short shank*. On an EIB Right board, the long shank of the EIB cable goes to connector 1, and the short shank goes to connector 2. Symmetrically, on an EIB Left board, the long shank connects to connector 2 and the short shank to connector 1. The cable connected to the EIB Right should be labeled XR. Similarly, the cable attached to EIB Left should be labeled XL.
3. Two gray SIB cables. These cables connect the EIB cables to the SIB. The SIB cable coming from EIB cable XR should be labeled XR, and for clarity, also labeled *To SIB XL*. The SIB cable from EIB cable XL should be labeled XL, *To SIB XR*.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	3	5	7	17	19	21	23	C	33	35	37	39	49	51	53	55
9	11	13	15	25	27	29	31	L	41	43	45	47	57	59	61	63
0	2	4	6	16	18	20	22	K	32	34	36	38	48	50	52	54
8	10	12	14	24	26	28	30		40	42	44	46	56	58	60	62

Figure 3. iWarp cell to board/slot mapping (64 cells).

## Attaching the EIB boards to the backplane

To gain access to the backplane, the power supply of the iWarp array should be removed. This is easily accomplished. First, slide out the supply on its extenders. Then, disconnect the following items: (1) the power plug; (2) a ground wire from the chassis; (3) a thin cable that runs to the clock board (though difficult to see, it unplugs from the power supply); (4) the four power cables from the chassis. Note that the red cables attach to the top, outside, and that the black cables attach to the bottom, inside. Once these cables have been disconnected, the power supply can be slid off of the extender slides by depressing the stops on the sides.

An aux-SIB must be inserted into one row (but not the first row, nor the last row of the array), between a left-hand cell and a right-hand cell. For example, let's connect the SIB to row 7 (counting from 1), between cells 48 and 55. From Figure 3, we see that cell 48 is on the quad-board in slot 14, and cell 55 is on the board in slot 17.

We must now identify the pathways that will be used to attach the SIB to the array cells. Figure 2 shows that each quad-board of the array has eight pathways going off-board: they are 1XL, 1XR, 2XL, 2XR, 1YL, 1YR, 2YL, 2YR. The X pathways connect the cells in the horizontal direction, and the Y pathways connect in the vertical.

We are only interested in attaching to a row, which implies using the X pathways. To find the appropriate pathways, note that Cell 48 occupies the upper left position of a quad-board, so its external pathways are indicated by 2/1. (See Cell 0 in Figure 2.) The second number tells us that the XL pathway of Cell 48 uses 1XL. Figure 4 shows that pins 1-19 of rows A and B are the 1XL pathway on the backplane. Cell 55 is on the upper right of a quad-board, and analogous to Cell 1, its XR pathway uses 1XR. From Figure 4, we see that 1XR uses pins 1-19 of rows E,F.

This means that we must remove the ribbon cable that runs from pins 1-19, rows E,F of slot 17 to pins 1-19, rows A,B of slot 14. (Note that, from the back, there are 6 vertical rows (columns?) of pins, lettered *right to left*, i.e., F E D C B A.) Also, remove the clock jumper that connects pin

PINS	Row F	Row E	Row D	Row C	Row B	Row A
1 - 19	1XR	1XR		13-17	1XL	1XL
21 - 39	1YR	1YR		33-37	1YL	1YL
41 - 59	2XR	2XR		53-57	2XL	2XL
61 - 79	2YR	2YR		73-77	2YL	2YL

Figure 4. Pin identification. Row C shows jumpers for clock.

C13 to pin C17 of slot 14. Remove a comparable jumper (C13 to C17) from slot 17.

The EIB Right must connect to an XR pathway, so it must attach to Cell 55, which is in slot 17. Thus EIB Right should be connected to pins 1-19 of rows E,F on slot 17. The EIB Left must connect to pins 1-19 of rows A,B of slot 14.

The EIB cables are connected to the EIB boards as indicated in the previous section.

## Booting the Array/Using the aux-SIB

To boot the iWarp array with an aux-SIB, first set the toggle switch on the aux-SIB to *SIB mode*. Apply power, or reset the aux-SIB so that it initializes its VMEbus interface. At this point, switch the aux-SIB to *cell mode* by flipping the toggle switch. The aux-SIB is now ready to be configured as a normal cell of the array. To accomplish this, a single-board computer (for example), acting as the VMEbus master, must write a 0x009e to the SIB's (16-bit) Global Control Register at location 0x7006 on the VMEbus, addressed in the short-address space. This bit pattern will enable the slave interface of the SIB to:

1. respond to both supervisory and non-supervisory bus cycles.
2. respond to VMEbus standard address cycles.
3. respond to extended address cycles
4. respond to the external clock from the array
5. reset the SIB, causing it to switch to the external clock.

It is convenient at this time to also set the Global Slave Address Register (GSAR, location 0x7004) to define the VMEbus address of the SIB's dual-ported memory. For example, to define the address as 0x10000000 (which must not conflict with any other devices on the VMEbus), write a 0x1000 into the GSAR.

The array is now ready to boot-strap the run-time system *iwrts*, and accept user programs. For the example described above, the aux-SIB is referenced as cell 76, and it may be treated as any other array cell, except that it has *no Y (up/down) pathways*.

The user should be aware of one detail when attempting to use the VMEbus master interface of the aux-SIB. Because it is treated as an ordinary cell during boot-up, the aux-SIB is configured exactly like every other cell. In particular, its memory size register is set to a value that disallows access to its control registers, which are located above 0xC00000. The LMSIZE register on the SIB must be reset if the user wishes to use these registers (which are used to control the SIB's VMEbus interface). This can be done by including the following lines in the aux-SIB code:

```
#include <regnums.h>
#include <asm/gen_asm.h>

asm_writereg(CSR_LMSIZE, 0x17f);
```

## Diagnostics

Upon power-up, or array reset, the aux-SIB should behave as a normal array cell after the toggle switch has been set to *cell mode*. Even in the absence of a software load, the bare hardware should execute an on-board ROM reset. The correct behavior is that all the cells will flash their lights, and then show a steady green light, and a flashing yellow light for a few seconds. All lights should then extinguish. The aux-SIB should participate in this process. If the low-level hardware does not respond in this manner, there is something fundamentally wrong.

Diagnostic behavior to look for:

1. When the array boots, the first thing that happens is that the SIB starts a global reset. Does the aux-SIB go into reset at this time?
2. Does the aux-SIB blink the green and yellow lights with the rest of the array?
3. Does the aux-SIB blink in time with the array?
4. Are the array cells to which the aux-SIB is attached both blinking in time with the array, or is one (or both) not blinking or blinking at a different rate?

These questions deal with the clock signal the SIB is receiving, and in turn re-generating to the array. If all the cells are not using the same clock signal, nothing will work.

If there is a problem, try the following.

First, be sure that a cell from the left-hand side of the array is connecting to the XR connector on the SIB, and the cell from the right-hand side of the array connects to the XL connector on the SIB.

Check the cables going from the iWarp array to the aux-SIB by removing the second SIB from the configuration file, and making a "long backloop" where the SIB used to be. Boot the array and run a few programs. (The *pwt* diagnostic program is a good choice.) It is not sufficient to simply boot; you must be able to boot and run programs and get output. The backloops are not used for booting (except for the master SIB), but they are used by the RTS for loading and running programs. It is possible to boot without the backloop, but the array will hang when executing a program.

Using this procedure, test both gray EIB cables, one at a time. This will verify that the differential driver boards and cables work as expected.

Be sure that the EXTCLK bit in the GCR register has been set. Check that the jumpers on the aux-SIB have been configured correctly. Be certain that the configuration file */iwarp/etc/cfg/host.0* correctly indicates the position of the aux-SIB. Note that this configuration file is read only once by the iWarp monitor demon at start-up. Consequently, any change to the configuration file will not take effect until the old demon is removed, and a new one started.

## REFERENCES

1. Bernecky, W.R., *Interface between Two Parallel Computers*, TM 941020, NUWC Detachment New London CT, 1994.
2. Furnanz, L., Kung H.T., et al, *iWarp MacroArchitecture Specification*, iWarp Tech Pub, INTEL Corp., Hillsboro OR, 1991.
3. Manseau, D. A., *iWarp Sun Interface Hardware External Product Specification*, INTEL Corp., Hillsboro OR, 1989.
4. Munoz, J. L., Bernecky W. R., *Invention Disclosure for Mapped Memory Interface for Communication between Multiple Computers*, NUWC Detachment New London CT, 1993.
5. Peterson, W., *The VMEbus Handbook*, VMEbus Int'l Trade Assoc, Scottsdale AZ, 1989.

**DISTRIBUTION LIST****Internal Distribution****Code**

0261	(NL Library (2))
0262	(NPT Library (2))
2094	(J. DePrimo)
2121	(W. Michael, J. Nuttal, M. Schindler)
2122	(T. Briggs, E. Medeiros, J. O'Sullivan)
2123	(A. Bernier, G. Bowman, R. Choma, S. Dzerovych, J. Ianniello, J. Law, M. Maguire, N. Owsley, G. Swope, T. Tetlow)
2133	(H. Schloemer)
2141	(P. Davis, R. Elswick)
2142	(D. Abraham)
2143	(J. Marsh)
215	(W. Coggins)
2151	(T. Choinski, D. DaRos, D. Organ)
2152	(I. Ferber, R. Latourette, R. Murdock)
2153	(W. Bernecky (4), S. Harrison, R. Howbrigg, J. Ionata, L. Karasevich, M. Krzych, J. Munoz, H. Watt, G. Zvara)
2154	(G. O'Brien)
2191	(B. Buehler, K. Collins)